Forgetting Reinforced Cases

Houcine Romdhane & Luc Lamontagne

Departement of Computer Science and Software Engineering
Laval University, Québec, Canada, G1K 7P4
{houcine.romdhane, luc.lamontagne}@ift.ulaval.ca

Abstract. To meet time constraints, a CBR system must control the time spent searching in the case base for a solution. In this paper, we presents the results of a case study comparing the proficiency of some criteria for forgetting cases, hence bounding the number of cases to be explored during retrieval. The criteria being considered are case usage, case value and case density. As we make use of a sequential game for our experiments, case values are obtained through training using reinforcement learning. Our results indicate that case usage is the most favorable criteria for selecting the cases to be forgotten prior to retrieval. We also have some indications that a mixture of case usage and case value can provide some improvements. However compaction of a case base using case density reveals less performing for our application.

Keywords: Case base management, Reinforcement learning, Anytime CBR.

1 Introduction

Games are an interesting laboratory for CBR as they involve interactivity, complex situations and evolving scenarios. As games often depict complex environments for which an exact model is difficult to build, they present some opportunities for the insertion of approximate approaches relying on example-based reasoning. Our work pertains to the application of CBR to games involving uncertainty and real-time response where time available to make a decision might vary with the context. Temporal constraints in games might be resulting from a change of level, a situation presenting increased complexity or a variation of tempo in the occurrence of events.

Computational time in a CBR cycle depends on the time dedicated to retrieval and adaptation. In a nearest neighbor setting, retrieval time depends on the number of cases being considered for making recommendations. Hence to meet time constraints a CBR system would either have to forget cases prior to case retrieval or to filter cases while performing retrieval. For this work, we adopt a forgetting approach to limit retrieval time. Characterization of adaptation time is more complex as this phase is less standardized. As most applications do not require adaptation or make use of simple case modification heuristics, we assume that most of the overall CBR computational time depends on the number of cases being considered during retrieval.

Forgetting cases involves removing those less valuable to the task being accomplished in order to maintain run-time performance and competence to an

acceptable level. Means are required to assess the value of individual cases. As most games are sequential in nature, reinforcement learning [15] is an interesting framework for determining the contribution or payoff expected from each case for solving some problems. Recent progress has been made in this direction [1, 2, 3, 9, 11] and we adopt this approach to assign values to individual cases.

In this paper, our goal is to determine if values obtained through reinforcement learning could provide a good indication on which cases a CBR system should forget to reduce the size of a case base. We conduct this work as a case study using Tetris, a game with simple rules presenting relevant time constraints. In section 2, we briefly survey the related CBR techniques used to manage a case base. Section 3 describes our research motivations and section 4 presents the reinforcement learning scheme we used for training a case base. In sections 5 and 6, we compare reinforcement value to other criteria such as case usage and case density to assess its importance as guidance for case forgetting. We also present in section 7 some experiments conducted with variants of a case deletion strategy proposed in [13] to determine if the compaction of case base is relevant to our problem.

2 Related Work

Managing the size of case bases has been addressed mostly by the community of CBR researchers working on case base maintenance strategies. Markovitch and Scott [6] showed that additional knowledge can degrade the performance of learning system and that removing some of this knowledge can help to correct the situation. Minton [7] proposed a deletion technique for explanation based learning systems. Racine and Yang [8] proposed to reduce the size of a textual case base by exploiting redundancy and incoherence relationships between cases.

Smyth and McKenna [13] proposed a technique for constructing compact case bases. Their approach relies on the notions of coverage and reachability [12], two criteria for estimating the competence of a case base when case adaptation is possible. This work highlights the importance of finding a trade-off between the competence and the efficiency of a CBR system. This is in opposition to machine learning deletion approaches mostly concentrating on computational time reduction.

It is important to mention that an alternative to removing cases prior to retrieval would be to filter cases during retrieval, and hence avoiding an exhaustive search of the case base. Schaaf [10] proposed *Fish and Shrink*, an approach where the similarity between two cases depends on how they are related to other cases already visited during retrieval. Also *Footprint-Based Retrieval* [14] is a retrieval procedure where a search is guided by a limited number of cases (the footprint set) covering all of the other cases in the case base. Following the selection of a reference case from the footprint set, the search is extended to covered cases, i.e. those that can be solved following adaptation of the reference case.

The main specificity of our work is that we make use of reinforcement learning to guide the forgetting of cases. Reinforcement learning (RL) in CBR has recently been studied by some authors. Gabel and Riedmiller make use of CBR to approximate RL versions of temporal difference learning [2] and Q-Learning [5]. Sharma et al. [11]

proposed an approach for conducting RL/CBR in games. In our previous work [9], we experimented with Q-Learning for evaluating existing case bases. Finally Aha & Molineaux [1] proposed a model for the reinforcement of continuous actions in CBR systems.

3 Motivations

As mentioned previously, we are interested in the application of CBR to games where time to make a decision varies for different situations. To conduct our study, we make use of Tetris [9]. Tetris consists of placing a dropping piece onto a set of columns to complete rows and avoid accumulation of pieces (Figure 1). Seven different shapes of pieces exist in the game. Placing each of them involves various combinations of rotation and sliding. This game presents interesting time constraints as a decision must be made before the piece touches the upper row of cubes. Time constraints in Tetris are caused by the following:

- As a piece is dropping, time is limited to make a decision.
- As the upper row of cubes rises up, space is reduced to rotate and translate the dropping piece.
- As the level of the game increases, the dropping speed increases.

This has an impact on the number of cases that can be processed to select a move. For instance, in most implementations of the game, a dropping figure takes less then a second for going down one step at level 0. And for each additional level, the time allocated for each step is reduced by tens of milliseconds. For our current implementation, this means that less than 25 000 cases could be explored for each step at level 0 during the retrieval phase. And each subsequent time reduction would corresponds to approximately 1000 cases that could not be used by the CBR component to meet its constraints.

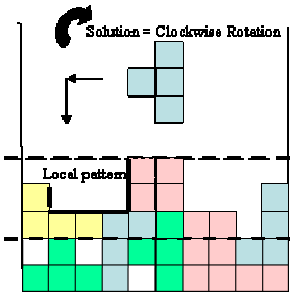

Figure 1 - Playing Tetris with CBR local patterns (from [9]).

We assume for the moment that adaptation efforts are limited. In our current setting, adapting a solution consists of rotating and translating the dropping figure. This can be easily accomplished by comparing cases and determining where some target patterns are located on the surface of the board (for additional details, see

details in [9]). As this computation is negligible compared to retrieval time, temporal constraints would not influence how adaptation is performed.

We are interested to determine which cases should not be considered by the CBR system when time is limited. To do so, we need some selection criteria to momentarily forget some of the cases. Such criteria would be associated to a performance profile indicating how the removing cases impact on the competence of a CBR system. As mentioned in section 2, criteria such as usage, recency and density could be considered. However, Tetris is a sequential game where a decision on how to place a figure impact on the next moves that can be done. This suggests that it is possible through reinforcement learning to assign a value to each case that would indicate the payoff expected from its usage. As reported in the next section, we made use of a Q-Learning scheme to determine such case values.

Given some profiles on how forgetting cases impact on CBR performance, we would need a framework to manage how to exploit these while reasoning online. Work on anytime reasoning could constitute an interesting framework to explore this issue. Case based reasoning could be formulated as an interruptible algorithm that can provide answers without running through a full completion of the reasoning cycle. As soon as one case has been consulted, the CBR system could be interrupted and return a solution whose quality might be partially satisficing. Quality measures are then required to guide a CBR component to keep searching for the amount of time being given. So our current goal is to learn how to characterize these performance profiles for CBR systems.

In the next sections, we will explore how reinforcement of learning is used to evaluate the payoff of a group of cases. Then we conduct a case study for evaluating the influence of various criteria on the performance of a CBR component for the game of Tetris.

4 Evaluation of Cases through Reinforcement Learning

Reinforcement learning [15] is a practical approach to learn consistent evaluations from observed rewards for complex Markov decision processes (MDPs). RL can be used to estimate the value of a state through temporal difference techniques or to evaluate state-action pairs through Q-Learning. For our application, we adopt a Q-Learning approach to evaluate a case base and the general training procedure is described in Figure 2.

Training relies on conducting multiple problem solving episodes with the CBR component. The case-based reasoning cycle (function CHOOSE-CBR-MOVE) used for this work contains the usual phases of CBR, i.e. retrieval and reuse. For our Tetris application, retrieval is performed by matching pattern cases with a subset of the columns on the board [9]. A pattern is represented as the heights of a sequence of N columns where a specific piece type can be dropped. This process returns the k best matching patterns. Case selection (function SELECT-BEST-CASE) is based on the value assigned to a case and returns, during exploitation, the most valued neighboring case. Case evaluation is performed using a Q-Learning procedure and is explained in the next paragraphs. Finally case adaptation (function ADAPT-SOLUTION) in Tetris is

done by modifying the orientation of a dropping piece. This can easily be determined by placing the piece where a pattern was matched and by applying the move (i.e. the rotation) recommended by the selected case.

```
procedure EVAL-CB-WITH-QLEARNING (CB)
   inputs:  CB, the case base used by the CBR cycle.
   local variables:
        State, the current state of the game (i.e. the height of the columns, initially empty).
        Problem, a problem to solve (i.e. a new piece P presented with orientation O)
        Case, one specific case (a pattern of columns + a piece + its orientation + a move)
        Previous_Case, the case recommended in the previous cycle.
        Solution, a move (i.e. the rotation and translation of the piece P).
        R, the reward obtained by applying Solution to State

   repeat
        Problem ← GENERATE-NEW-PROBLEM(); // Select randomly a piece to be placed
        Case, Solution ← CHOOSE-CBR-MOVE(Problem, State, CB);
        Previous_Case.Value ← UPDATE-CASE-VALUE (Previous-Case, Case, R)
        R, State ← reward and new state resulting from the application of Solution to State
        Case.usage ← Case.usage + 1
        Previous-Case ← Case
   until some stopping criterion is satisfied

function CHOOSE-CBR-MOVE(New-problem, State, CB)  returns a case and a solution
  local variables:
          k, the number of nearest neighbors being considered
          Case, one specific case (a pattern of columns + a piece + its orientation + a move)
          Candidates, some similar cases
          New-solution, a move (i.e. the rotation and translation applied to the piece P).
  // Retrieval
  Candidates ← FIND-KNN(k, New-Problem, State, CB)
  // Reuse
  Case ← SELECT-BEST-CASE(Candidates, New-Problem, State)
  New-solution ← ADAPT-SOLUTION(Case, New-Problem, State) // rotate and translate the piece
  return Case, New-solution
```

Fig. 2 - CBR problem-solving cycle for the reuse of reinforced cases with its interpretation for the game of Tetris.

The training procedure (EVAL-CB-WITH-QLEARNING) for evaluating cases provided by a case base goes as follows. We start with some initial evaluations *Value* corresponding to the rewards assigned by Tetris to each of the cases of the case base. Then we let the CBR component play games during which cases are selected and modified. For a case C_t selected at time t, a revision of its value is performed by UPDATE-CASE-VALUE using the following function:

$$Value(C_t) = (1-\alpha)\,Value(C_t) + \alpha\left(R(C_t, state_t) + \gamma \max_{C_{t+1}} Value(C_{t+1})\right) \quad \textbf{(1)}$$

where R is the reward obtained by applying the move adapted from C_t to the new target surface at time t. The discount factor γ assigns some importance to future moves and α determines the trade-off between the current value of a case and its potential future payoff.

In the update equation (1), C_{t+1} corresponds to the case selected by the CBR system at the iteration $t+1$. This captures the idea that an efficient CBR system should seek the maximum payoff expected from future moves. Hence the value of future moves should be backed up in the value of C_t. As the CBR cycle always chooses the most valued case present in the case base (function SELECT-BEST-CASE), we assume that the next selected case C_{t+1} is a good approximation of the maximum solution to be applied to $state_{t+1}$. From an implementation point of view, the value of C_t is updated during the CBR cycle at time $t+1$.

In order to prevent falling into local optima regions, the training process is allowed to explore the search space by selecting non maximal cases from the set of nearest neighbors. This is captured by a softmax rule [16] where the probability of selecting one of the nearest neighbors is given by

$$P(case) = \frac{e^{Value(case)/\tau}}{\sum_{case_i \in knn} e^{Value(case_i)/\tau}} \quad (2)$$

where τ is an exploration factor (or temperature). This factor is reduced progressively with time to bring the training algorithm to adopt a greedy exploitation behavior (i.e. select the most valued case). And exploration is not allowed during the exploitation of the case base once training is completed.

5 Forgetting Cases using Reinforcement Values

The first issue we addressed was to determine if reinforcement learning helps to provide good indications on how to reduce the size of a case base. We conducted an experiment with an initial case base of 55 000 cases trained with Q-Learning and softmax exploration as described in the previous section. All the results were obtained using the Tieltris testbed and we made use of the rewards assigned by this implementation of Tetris to update the value of the cases.

The two following parameters for each case were determined following the training session:

- Case value (V) : the reinforcement value resulting from successive updates of a case during training;
- Case usage (U): the number of times a case was selected during the training experiment.

For evaluating the impact of these two criteria on the performance of the CBR component, we built performance profiles as follows:

- We impose a threshold value on one of the criteria and we remove all cases presenting lower characteristics;

- We play a number of games to evaluate the performance obtained by using the remaining cases. In our experiments, 100 games were played for each trial. And for each game, the pieces to be dropped were selected randomly.
- We increase the cutoff threshold and repeat this evaluation until the case base gets empty.

Performance profiles for both criteria are presented in Figure 3. Performance is characterized by the number of lines removed during a Tetris game. We obtained similar results for indicators such as game scores and the number of moves played during a game. Hence these are not presented in this experimentation section as they would lead to similar conclusions.

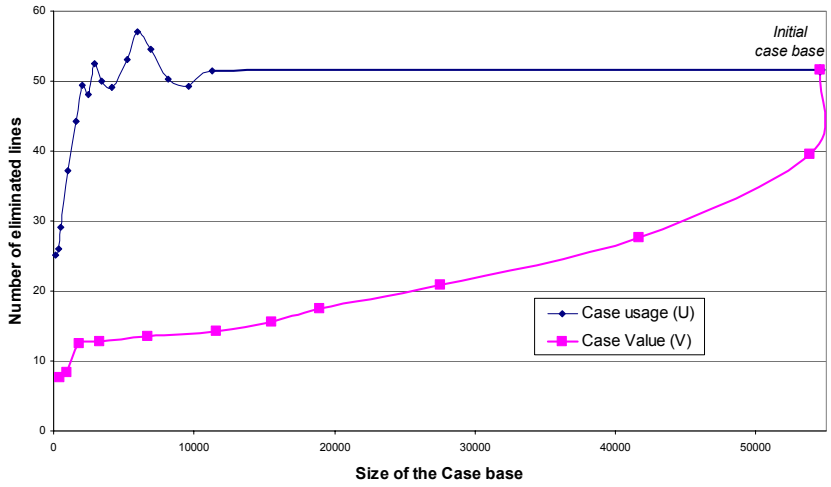

Fig. 3. Performance profiles for case usage and case reinforcement value. These are performance profiles obtained by progressively removing cases from an initial case base of 55 000 cases.

Our experimental results indicate that progressively removing cases based on their reinforcement value prematurely yields an important degradation of performance (see curve *V*). By deleting about one thousand of the least valued cases (i.e. 2% of the initial case base), the CBR component removes on average 25% fewer lines during a game. We conjecture that some of these cases are applied to difficult moves offering little payoff. These states might often be visited during the course of a game and, by removing them from the case base, the CBR system is left with no guidance on how to manage these situations. Then by progressively removing additional cases, we notice a performance degradation which is almost linear. Finally, the system becomes totally inefficient when less than a thousand cases are used by the system.

Case usage presents a much different performance profile. The number of lines being removed remains constant even after removing 40 000 of the less frequently used cases (i.e. 2/3 of the initial case base). However with a CBR system operating

with between 2000 and 12 000 cases, the performance starts to oscillate. In this transition phase, we notice some peak performance even surpassing results obtained from larger case bases. However no guarantee for improvements can be made due to the instability of the performance profile in this region. Finally, as we keep removing the last few cases (< 1000), the performance drops rapidly to lower values as the number of cases becomes insufficient.

Hence, from this experiment, we can conclude that case usage (U) provides a better decision criterion for progressive forgetting of cases for our application. Results also indicate that reinforcement value (V) is not by itself an informative criterion for reducing the size of the case base. It is also interesting to note that the size of the CBR memory can significantly be reduced without impacting severely on its performance.

6 Combining Multiple Criteria

Based on the results of the previous experiment, we tried to combine both criteria to see if additional improvements could be obtained using a mixture of them. We considered 2 variants:

- Additive form: a linear combination of the two criteria.

$$f(Case) = \alpha\ Case.U + \beta\ Case.V$$

- Product form: a product of the two criteria where V is normalized to remove negative reinforcement values.

$$f(Case) = \alpha\ Case.U \times (Case.V + \beta)$$

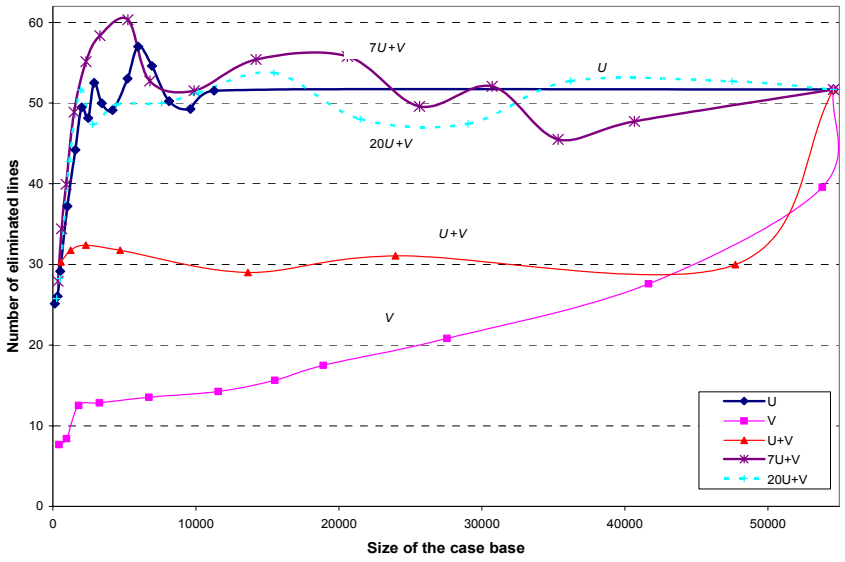

Fig. 4. Combining criteria U and V for forgetting cases.

Figure 4 displays some experimental profiles we obtained for these forms. For most of the coefficients α and β, making a linear combination of *V* and *U* represents a compromise between these two criteria. Hence the resulting combination generates a performance profile that stands between those of the individual criteria. For instance, the curve *U*+*V* is a slight improvement over *V* but does not nearly surpass the performance profile of *U*. And as the component *U* becomes more important in the mixture, the performance profile of the combination resembles the profile of the case usage criteria. We tried different linear combinations and we found that, for our Tetris application, the best results were obtained when $\alpha/\beta \approx 7$. While some significant improvements can be expected from this combination, we can not however conclude that such a linear combination guarantees better results for any number of cases. However it seems to be an advantageous option for case bases comprising less then 20 000 cases.

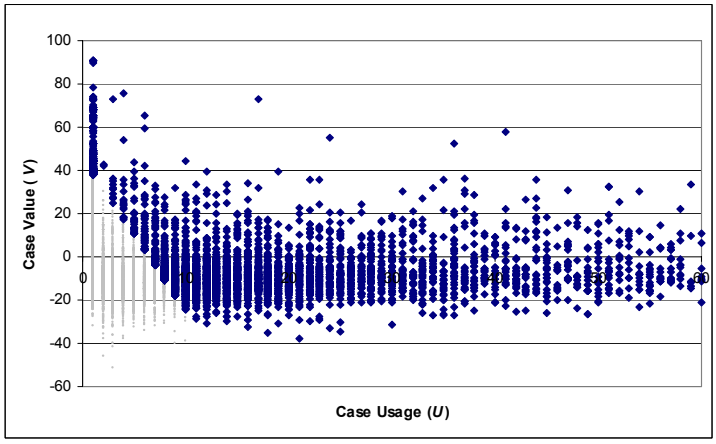

Fig. 5. Case being removed from the case base for a linear combination of $7U + V$. Points with a lighter shade of gray represent cases that have been removed from the case base.

Figure 5 illustrates the composition of the case base when reduced to 5200 cases (i.e. less than $1/10^{th}$ of its initial size) by applying a linear combination of $7U + V$ (i.e. a ratio of $\alpha/\beta = 7$). If we refer to Figure 4, this corresponds to the optimum peak value obtained by this performance profile. The dark points correspond to those cases left in the case base while the gray points correspond to those being removed. We notice that while more than 90% of the cases have been removed, those are highly concentrated in the bottom left part of the figure. This region depicts cases with both lower usage and reinforcement values. This suggests that this is a region where it is beneficial to concentrate its forgetting efforts.

Finally a comparison of the additive and product forms of combination, as illustrated in Figure 6, clearly indicates that the product form does not provide any additional advantage over the best results obtained using an additive form.

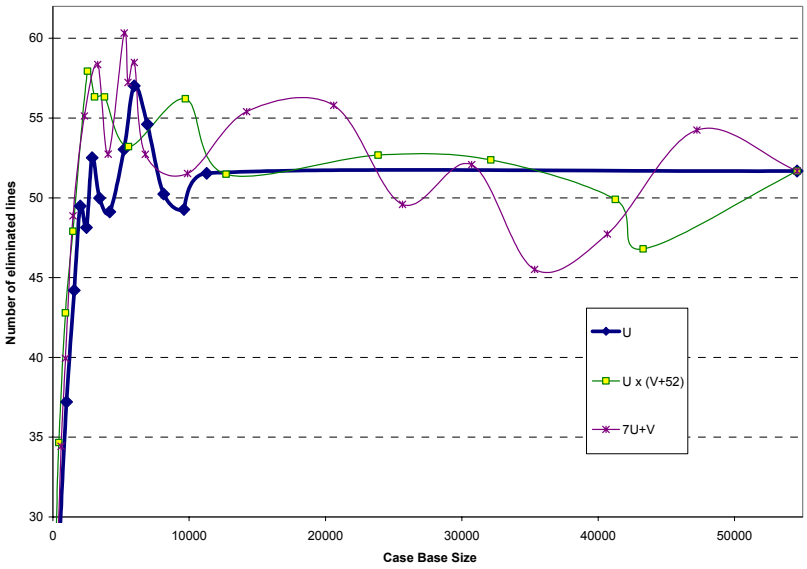

Fig. 6. Comparison of additive and product forms.

7 Compacting the Case Base

Smyth and McKenna proposed in [13] a case deletion strategy to build compact case bases. The basic idea is to select a subset of cases that perform well and that are well dispersed in the problem space. To achieve this, we evaluate all the cases of the CBR system with respect to some criterion and sort them in decreasing order. Then we create a new case base and we successively add cases, following a decreasing order, that can not be solved by other cases already present in the new case base. This algorithm is presented in Figure 7.

From an intuitive point of view, compaction seems to favor the removal of cases from different regions of the problem space. This approach should provide performance profiles different from those of the progressive forgetting studied in section 5 and 6 of this paper where cases were removed in concentrated areas. By diversifying the regions where cases are removed, we could expect a better preservation of the competence of the CBR system.

Authors in [13] proposed a competence criterion, based on reachability and coverage [12], to determine the extent of cases that can be solved. For our game of Tetris, we consider that a case can solve another if both pertain to the same type of piece and if the configurations of columns are sufficiently similar (i.e. superior to

some similarity thresholds). This is a reasonable assumption as, for similar board configurations, a piece can be rotated and translated to produce a reusable solution.

```
function CONSTRUCT_COMPACT_CB (Cases, Criterion), returns a case base
    inputs:  Cases, all the case originally used of the CBR system.
        Criterion, the attribute used to rank cases.
    local variables:
        Sorted_Cases, the cases ranked by some criterion
        New_CB, the result of this function, a subset of the original cases
        SIM_THRESHOLD, a cutoff threshold for forgetting cases.
        Sorted_Cases ← SORT-CB(Cases, Criterion);
        New_CB ← {};
        CHANGES ← true;

    while CHANGES do
        CHANGES ← false;
        For each case C ∈ Sorted_Cases
            if MAX-SIMILARITY (C, New_CB) < SIM_THRESHOLD then
                CHANGES ← true;
                Add C to New_CB;
                Remove C from Sorted_Cases;
    return New_CB

function MAX-SIMILARITY (C, Cases) returns a similarity value
    Sim = 0.0;
    for each case C' ∈ Cases do
        if C.type = C'.type;   // similarity is restricted to pieces of the same type.
        Sim ← MAX(Sim,GET-SIMILARITY(C, C'));
    return Sim;
```

Fig. 7. Algorithm for compacting a case base – Adapted from [13].

The function CONSTRUCT_COMPACT_CB is sensitive to the ordering of the original cases. To control this factor, we need to sort the cases with respect to some criterion (function SORT-CB). We compared experimental results obtained when cases were sorted based on their usage and reinforcement values. We also considered the possibility, for building a compact case base, to remove cases based on their similarity with other cases. So we estimated for each case the density of its neighborhood (i.e. the average similarity of its nearest neighbors) and we made use of this criterion for our experiments. The resulting performance profiles are presented in Figure 8. To obtain various sizes of case bases, we applied the compaction function for different values of similarity thresholds *SIM_THRESHOLD*.

We note that the compaction of a case base brings a constant and linear decrease for most of the sorting criterion. While this approach is superior to progressively forgetting cases based on reinforcement values, the various performance profiles obtained using U, V and case density as sorting criteria reveal largely inferior to the progressive approach with case usage.

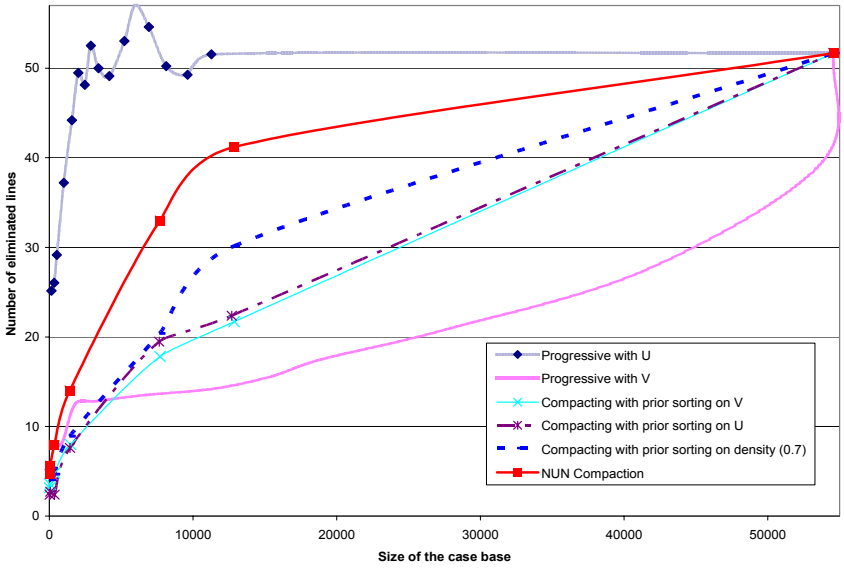

Fig. 8. Compaction of cases using various sorting criteria. Progressive performance profiles refer to the removal of cases without compaction as described in sections 5 and 6 of this paper.

Finally we also tried a variant of this approach called NUN [13]. While it compares favorably to the other compaction performance profiles, it does not outrank the results obtained by progressive forgetting with case usage.

8 Discussion

We learned from our case study with Tetris that profiles can be built for estimating the performance of a CBR system with various case base sizes. Our results clearly indicate that, for our application, case usage is the best criteria for forgetting cases. Reinforcement values are not necessarily by themselves good indicators for forgetting cases but could contribute if combined with other criteria. Finally a progressive reduction of the case base seems to be more appropriate than compaction for our application.

However this does not mean that reinforcement learning does not contribute to CBR. Our previous work [9] clearly illustrated that reinforcement training of a case base can improve the overall performance of a CBR system.

As mentioned in section 3 of this paper, one of our objectives is to make use of performance profiles to limit the size of a case base when facing time constraints (which we referred to as anytime CBR). This supposes that using fewer cases would imply a degradation of performance. But one of our findings is that using the 55 000 cases that were initially provided to build the CBR system, we could play Tetris at an

optimum level by using a subset of approximately 5 000 cases. Hence with such a limited number of cases, retrieval time would never exceed temporal constraints imposed by the game. This is a surprising result as our internal case representation for Tetris has a dimensionality of approximately 70 million different states. But it seems to be a particular situation where the competence of the CBR system is not proportional to the number of cases being used.

We expect that the proposed approach could be applicable to other time-constrained games. For instance, pursuit games could benefit from performance profiles and by dimensioning the size of the case base to meet time-critical situations. For instance, a reactive game like Pacman could be simple and interesting laboratory for conducting such a study. This remains to be done as future work.

References

1. Aha, D., Molineaux, M., (2008) Learning Continuous Action Models in a Real-Time Strategy Environment, *Proceedings of the Twenthy First International FLAIRS Conference*, AAAI Press, pp. 257-262.
2. Gabel, T., Riedmiller. M., (2005) CBR for State Value Function Approximation in Reinforcement Learning. *Proceedings of ICCBR'05*, Springer, pp. 206-220.
3. Gabel T., Riedmiller. M., (2007) An Analysis of Case-Based Value Function Approximation by Approximating State Transition Graphs*, Proceedings of ICCBR'07*, pp. 344-358.
4. Haouchine, K. M., (2006) Chebel-Morello, B., Zerhouni, N.: Méthode de Suppression de Cas pour une Maintenance de Base de Cas, *14e Atelier de Raisonnement à Partir de cas*.
5. Kira, Z., Arkin, R. C., (2004) Forgetting Bad Behavior: Memory Management for Case-Based Navigation, *Proceedings of the 2004 International Conference on Intelligent Robots and Systems*, IEEE, pp. 3145-3152.
6. Markovitch, S., Scott, P. D., (1988) The Role of Forgetting in Learning. *Proceedings of The Fifth International Conference on Machine Learning*, Morgan Kaufmann, pp. 459-465.
7. Minton, S., (1990) Qualitative Results Concerning the Utility of Explanation-Based Learning. *Artificial Intelligence*, no. 42, pp. 363-391.
8. Racine, K., Q. Yang, Q., (1997) Maintaining Unstructured Case Bases. *Proceeding of ICCBR'97*, Springer-Verlag, pp. 25-27.
9. Romdhane, H., Lamontagne, L., (2008) Reinforcement of Local Pattern Cases for Tetris, *Proceedings of Twenthy First International FLAIRS Conference*, AAAI Press, pp. 263-269.
10. Schaff, J., (1996) Fish and Shrink : A Next Step Towards Efficient Case Retrieval in Large Scale Cases-Bases, *Advances in Case-Based Reasoning*, LNAI 1168, Springer-Verlag, pp. 362-376.
11. Sharma, M., Holmes, M., Santamaria, J., Irani, A., Isbell, C., Ram, A., (2007) Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL; *Proceedings of IJCAI'07*, pp. 1041-1046.
12. Smyth, B., Keane, M., (1995) Remembering to Forget : A Competence Preserving Deletion Policy for Case-Based Reasoning Systems, *Proceedings of IJCAI-95,* Morgan Kaufmann, pp. 377-382.
13. Smyth, B., McKenna, E., (1999) Building Compact Competent Case-Bases. *Case-Based Reasoning Research and Development*, LNAI 1650, Springer-Verlag, pp. 329-342.
14. Smyth, B., McKenna, E., (1999) Footprint-Based Retrieval, *Case-Based Reasoning Research and Development*, LNAI 1650, Springer-Verlag, pp. 343-357.
15. Sutton, R., Barto, A., (1998) *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA.